Printer port hosts precision analog I/O board

Huw Jones, Gyrus Medical Ltd, Cardiff, Wales, UK

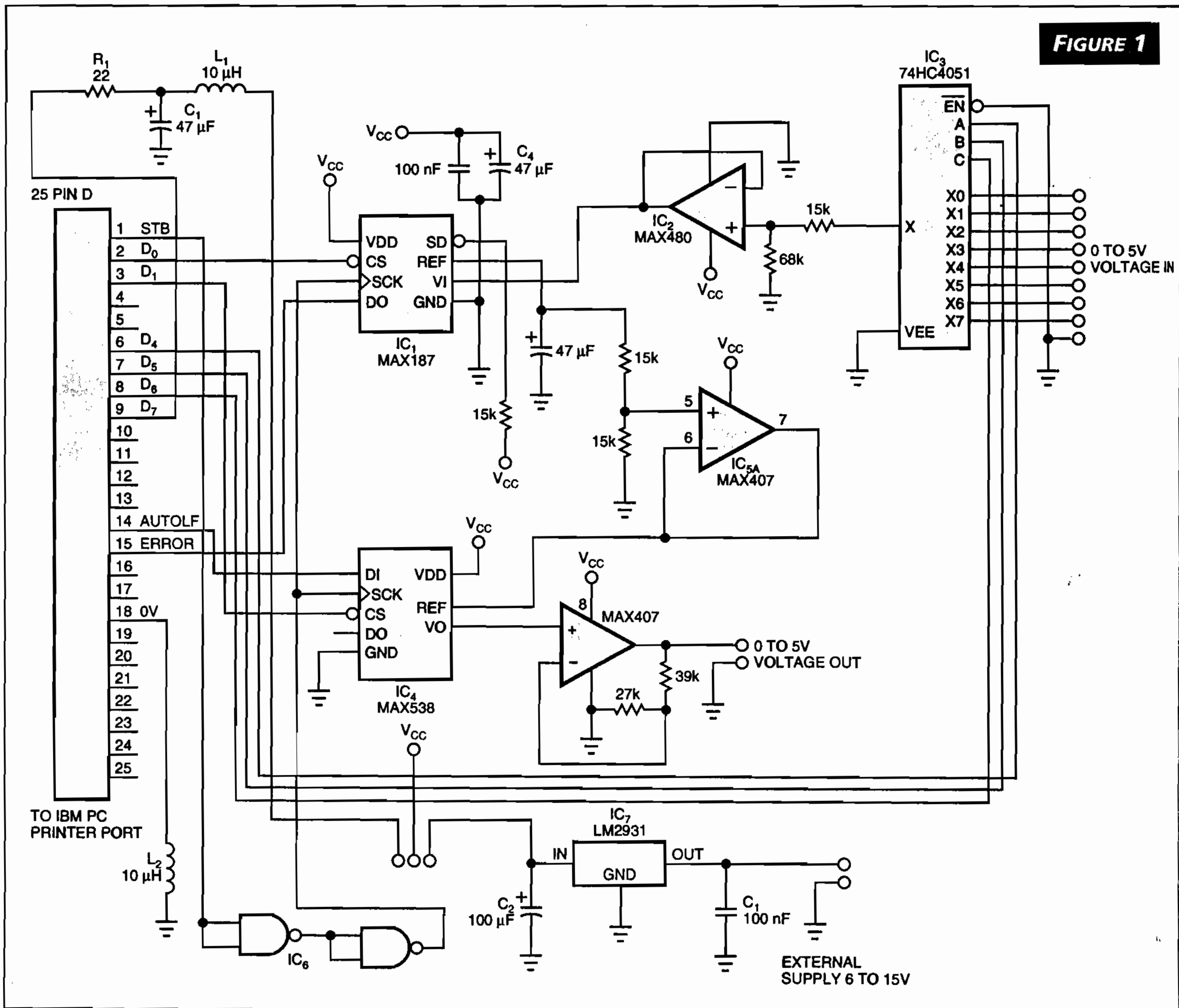
BBS The 12-bit analog I/O board in Fig 1 plugs into a PC's printer port. Thus, you can move the board around your laboratory more easily than you can exchange A/D boards that plug into the PC's backplane. The board handles eight 1-kHz input signals ranging from 0 to 5V max.

IC_1 is a serial, 12-bit A/D converter having an internal 4.096V reference and an internal track-and-hold circuit. Op-amp IC_2 provides a low-impedance source for IC_1. IC_2 has a V_{OS} of 70 μV, which is well within ½-bit conversion accuracy. Further, IC_2's rail-to-rail outputs come to within 1 bit of IC_1's full input range. However, the circuit's relatively slow slew rate limits input frequencies to below 1 kHz. Analog multiplexer IC_3 allows you to select any one of eight input channels.

D/A-converter IC_4 furnishes a 12-bit output. IC_4 derives its reference voltage from IC_1's reference output. Op-amp IC_{5A} and its associated components develop IC_4's 2.048V reference.

Schmitt-trigger IC_6 squares up the serial clock's edges (STB). This squaring up is a precaution and is, therefore,

FIGURE 1



This 12-bit analog I/O board plugs into a PC's printer port, allowing you to move the board around your laboratory easily.

unnecessary if your PC has HCMOS-compatible output lines. Also, depending on your particular PC, printer-port-signal D_7 provides 5V power via R_1, C_1, and L_1. Obtain the best performance, however, by using an external supply. Low-dropout-regulator IC_7 yields a stable 5V from a 6 to 15V input. Inductor L_2 reduces digital noise from the PC's ground rail.

Listing 1 is a sample interface routine written in C++. You can easily convert this listing to standard C. A/D-conversion speed depends entirely on software-execution speed. The ZIPfile attached to EDN BBS /DI_SIG #1647 contains the listing as well as a write-up. (DI #1647)

To Vote For This Design, Circle No. 346

LISTING 1—SAMPLE CONVERSION C++ ROUTINE

```
#define STB 0x01        /* STROBE on pin 1 */
#define ALF 0x02        /* AUTO LINE FEED on pin 14 */
#define INIT 0x04       /* INITIALIZE on pin 16 */
#define SELO 0x08       /* SELECT OUT on pin 17 */
#define IRQ 0x10        /* INTERRUPT ENABLE bit */
#define BUSY 0x80       /* BUSY input on pin 11 */
#define ACK 0x40        /* ACK input on pin 10 - this is inverted */
#define PAPER 0x20      /* PAPER ok input on pin 12 */
#define SELI 0x10       /* SELECT IN pin 13 */
#define PERROR 0x08     /* ERROR signal on pin 15 */
#define D00             // data line daclarations
#define D11
#define D22
#define D33
#define D44
#define D55
#define D66
#define D77
#define HIGH 1
#define LOW 0

enum onoff {high = 1, low = 0} ;   /* for bit control */

#include <dos.h>
#include <conio.h>
#include <iostream.h>

//*** read byte back from of selected printer data port ***
//* entry : printer port [1,2]
//* exit  : value read back from selected data port
char datain(int ppnum)
{
   int addr = 0x378 ;
   if (ppnum != 1)
       addr = 0x278 ;    // unless explicit port 1 , use port 2
   return(inportb(addr)) ;  // will be previously written data
}

//*** general purpose adjust control bit for selected printer port ***
//* entry : printer port [1,2], port bit as bitmask for control
//* exit  : control pin changed on selected port
void control_bit(int ppnum,char bitmask, onoff state)
{
   int addr = 0x378+2 ;
   char temp ;
   if (ppnum != 1)
      addr = 0x278+2 ;       // unless explicit port 1 , use port 2
   temp = inportb(addr) ;    // reads backe last control value from register
   temp ^= 0x0b ;            // bit inversion correction
   if (state == low)
     temp &= bitmask ;       // clear pin, no check for integrity
    else
     temp |= bitmask ;       // set pin high
   temp ^= 0x0b ;            // resurrect inversion
   outportb(addr,tamp) ;     // written
}

//*** general purpose adjust data bit for selected printer port ***
//* entry : printer port [1,2], port bit number
//* exit  : data pin changed on selected port
   void data_bit(int ppnum, int bitnum, onoff state)
{
   int addr = 0x378 ;
   char temp ;
   char bitmask ;
   bitmask = (1 << bitnum) ;  // derive mask
   if (ppnum != 1)
               addr = 0x278 ;      // unless explicit port 1 , use port 2
   temp = inportb(addr) ;     // reads back register
   if (state == low)
     temp &= bitmask ;        // clear pin, no check for integrity
    else
     temp |= bitmask ;        // set pin high
   outportb(addr,temp) ;      // written
}

//*** read status byte of selected printer port ***
//* entry : printer port [1,2]
//* exit  : value obtained from selected status port
char statin(int ppnum)
{
   int addr = 0x378+1 ;
   if (ppnum != 1)
       addr = 0x278+1 ;            // unless explicit port 1 , use port 2
   return(inportb(addr)) ;   // status available in bits 7-3
}

//*** write data byte to bits D0-D7 of selected printer port ***
//* entry : printer port [1,2], data to be written
//* exit  : selected port written
void dataout(int ppnum,char data)
{
   int addr = 0x378 ;
   if (ppnum != 1)
       addr = 0x278 ;              // unless explicit port 1 , use port 2
   outportb(addr,data) ;     // written
}

//*** set up a/d input channel selection on D6-D4 ***
//* entry : port 1/2 and channel in range 0 to 7
//* exit  : HC4052 multiplexer routed
void mux(int port,int chan)
{
   char temp ;
   temp = datain(port) ;
   temp &= 0x8f ;              // mask out bits 4 through 6
   temp |= (chan << 4) ;
   dataout(port,temp) ;
}

//*** control synthesised Vcc rail powered from D7 ***
//* entry : port 1/2 and condition
//* exit  : power rail enabled/disabled with time lag
void power(int port, onoff state)
{
   if (state == high)
       data_bit(port D7,high) ;   // enable 5V
     else
       {
       control_bit(port,STB,low) ;  // set all control lines at 0V before
       control_bit(port,ALF,low) ;  // removing 5V supply
       mux(port,0)      ;           // same for data lines
       data_bit(port D0,low) ;
       data_bit(port D1,low) ;
       data_bit(port D7,low) ;     // now switch off 5V
       }
   delay(200) ;
}

//*** perform a serial clock pulse ***
//* entry : port 1/2, SCLK low
//* exit  : one clock pulse L->H->L generated
void clockbit(int port)
{
   control_bit(port,STB,high) ;   // force serial clock line high
   control_bit(port,STB,low) ;    // now low
}

//*** read 12 bit a/d input via MAX187 ***
//* entry : port 1/2, CS for a/d high,serial clock line low
//* exit  : 12 bit value in range 0-4095
int read_adc(int port)
{
   int clocks ;
   int adin = 0 ;
   int temp,timeout ;
   data_bit(port D0,low) ;     //chip select MAX187 via D0 going low
   timeout = 1000 ;
   do
   {
     temp = statin(port) & PERROR ;  // wait for busy to complete
     timeout-- ;
   }
   while ((!temp) && (timeout)) ;  // ready when DO goes low to high

   for (clocks = 0 ; clocks < 12 ; clocks++)    // read 12 bit data plus
     {                                          // one redundant
       adin <<= 1 ;           // adjust for msb bit shift out from MAX187
       clockbit(port) ;            // toggle serial clock
       if (statin(port) & PERROR)
         adin |= 0x01  ;           // shift in a 1 bit
     }
  clockbit(port) ;            // final redundant clock to complete
  data_bit(port D0,high) ; // disable MAX187
  return(adin) ;
}

//*** write 12 bit d/a via MAX538 ***
//* entry : port 1/2, voltage out value (0-4095) , CS for d/a high
//* exit  : d/a updated
void write_dac(int port,int dacval)
{
   int clocks ;
   int mask = 0x0800 ;
   control_bit(port,ALF,low) ;  //set data low
   data_bit(port D1,low) ;     // chip select MAX538 via D1 going low
   for (clocks = 0 ; clocks < 12 ; clocks++)   // write 12 bit data
     {                                      //set data on AUTO LINE FEED
      (dacval & mask) ? control_bit(port,ALF,high) : control_bit(port,ALF,low)
        ;
      clockbit(port) ;         // toggle serial clock
      mask >>= 1 ;      // realign mask for msb first operation
     }
   data_bit(port D1,high) ;   // disable MAX538
}

//*** test out main routine ***
main()
      {
   int port ;
   int i,temp ;
   int dac,adc ;
   port = 1 ;                // assume LPT1:
   power(port,high) ;         // enable 5V rail
   mux(port,0) ;             // select a/d channel 0
   clrscr() ;
   dac = 0 ;
   while (! kbhit())         // loop until key press
   {
     write_dac(port,dac) ;  // 12 bit d/a saw tooth output
     dac++ ;
     if (dac > 4095)
       dac = 0 ;
     adc = read_adc(port) ;  // read 12 bit a/d
     gotoxy(10,5) ;
     cout << "a/d input = " << adc / 1000 << '.' << adc % 1000 << "V   " ;
   }
   power(port,low) ;           // disable 5V rail
}
```